

# Osnovi digitalne elektronike IR

vežba 2

Odsek za elektroniku

Elektrotehnički fakultet,  
Univerzitet u Beogradu

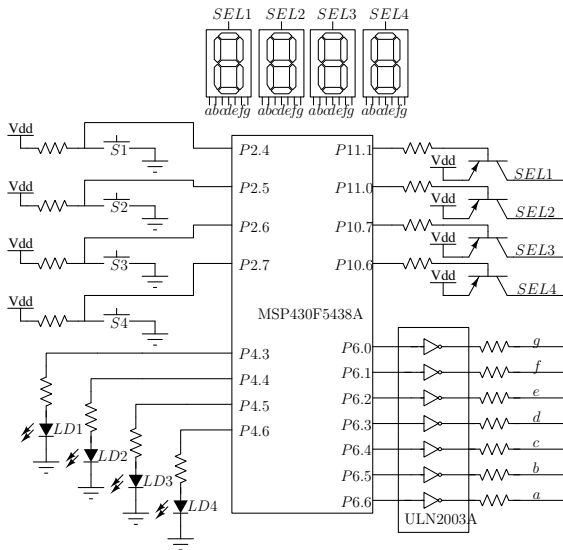
2018/2019

- 1 Razvojno orkuženje
  - Hardversko razvojno okruženje
    - Portovi
  - Softversko razvojno okruženje
    - *Include* fajlovi
    - Linkovanje
    - Kompajliranje
    - Debugovanje
- 2 Primeri
  - Rad sa diodama
  - Rad sa tasterima
  - Prekidi
  - Sedmosegmentni displej

# Pregled

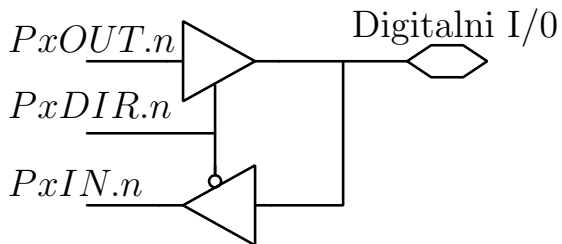
- 1 Razvojno orkuženje
  - Hardversko razvojno okruženje
    - Portovi
  - Softversko razvojno okruženje
    - *Include* fajlovi
    - Linkovanje
    - Kompajliranje
    - Debagovanje
- 2 Primeri
  - Rad sa diodama
  - Rad sa tasterima
  - Prekidi
  - Sedmosegmentni displej

# Opis hardvera



# Digitalni portovi

Svaki port se kontroliše sa četiri registra



## Registri za kontrolu portova 1/2

PxIN - ulazni registar ( $x = 1 - 11$ )

- očitani bit = 0 - na ulazu je nizak logički nivo
- očitani bit = 1 - na ulazu je visok logički nivo

PxOUT - izlazni registar ( $x = 1 - 11$ )

- upisani bit = 0 - na izlazu je nizak logički nivo
- upisani bit = 1 - na izlazu je visok logički nivo

PxDIR - registar selekcije ulaza ili izlaza ( $x = 1 - 11$ )

- upisani bit = 0 - pin je ulazni
- upisani bit = 1 - pin je izlazni

PxSEL - registar selekcije alternativnih funkcija ( $x = 1 - 11$ )

- upisani bit = 0 - ulazno/izlazna funkcija pina

## Registri za kontrolu portova 2/2 (Registri kontrole prekida)

### PxIFG - registar flegova ( $x = 1, 2$ )

- očitani bit = 0 - na pinu se nije dogodila promena koja izaziva prekid
- očitani bit = 1 - na pinu se dogodila promena koja izaziva prekid

### PxES - registar selekcije ivice ( $x = 1, 2$ )

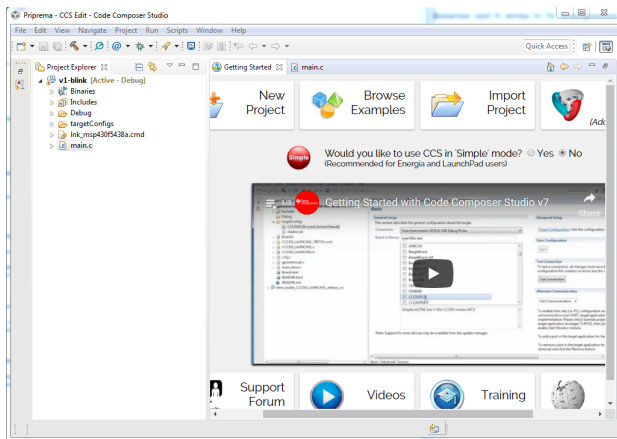
- upisani bit = 0 - prekid se događa na rastuću ivicu signala na pinu
- upisani bit = 1 - prekid se događa na opadajuću ivicu signala na pinu

### PxIE - registar dozvole prekida ( $x = 1, 2$ )

- upisani bit = 0 - prekid nije dozvoljen za promenu stanja na pinu
- upisani bit = 1 - prekid je dozvoljen za promenu stanja na pinu

# Code Composer Studio V7

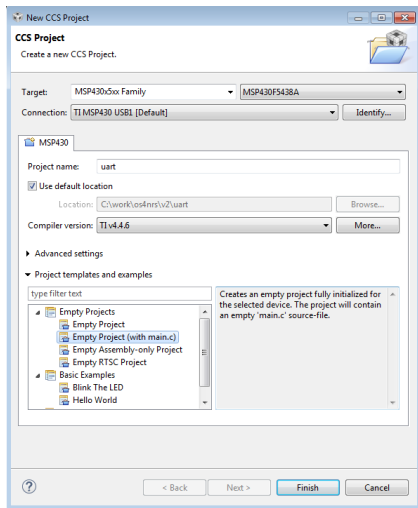
Preuzimanje softvera moguće sa linka  
<http://www.ti.com/tool/CCSTUDIO>





# Kreiranje projekta

S obzirom da je tražena implementacija u C-u, u dijalogu kreiranja projekta bira se C-ovski template



# Osnovni fajl

Predefinisani template za C-ovski projekat prikazan je na slici

The image shows the CCS Edit IDE with the main.c file open. The code in main.c is as follows:

```

1 #include <msp430.h>
2
3 /*
4  * main.c
5  */
6 int main(void) {
7     WDTCTL = WDTPW | WDTHOLD; // Stop watchdog timer
8
9     return 0;
10 }
11

```

The assembly template (asm.asm) is also shown, with the following key sections:

```

1 .def _reset
2
3 .include C:\TISSP\msp430.h
4
5 .def _reset
6     .export program_entry_point to
7     .make it known to linker
8
9     .start
10    .assemble into program memory
11    .override ELF conditional linking
12    .and retains current section.
13    .and retains any sections that have
14    .references to current section.
15
16
17
18
19 _reset
20     mov.w #_STACK_END,sp
21     mov.w #0,WDTPW,WDTHOLD // Stop watchdog timer
22
23
24 .Main loop here
25
26
27
28
29
30
31 Stack pointer definition
32
33 .global _STACK_END
34 .sect ".stack"
35
36
37 Interrupt vectors
38
39 .sect ".reset"
40 .short _reset

```

Annotations in the image explain the following elements:

- Struktura projekta:** Points to the Project Explorer showing the file structure.
- Fajl sa opisom hardverskih specifičnosti izabranog mikrokontrolera:** Points to the `#include <msp430.h>` line.
- Osnovna main funkcija programa:** Points to the `int main(void) {` line.
- Isključivanje Watchdog tajmera:** Points to the `WDTCTL = WDTPW | WDTHOLD;` line.
- Vrednost koju funkcija vraća:** Points to the `return 0;` line.
- Za razliku od assemblera u C-u se sva inicijalizacija obavlja automatski - od strane prevodoca. To podrazumeva između ostalog da prevodilac brine o postavljanju koda u memoriju, postavljanju reset prekida, kao i inicijalizaciji SP-a:** A large text box explaining the role of the compiler in initialization.

## Include fajl

Korišćenje pojmova definisanih u *include* fajlu olakšava kodiranje ali i kasniji prelaz sa jednog na drugi mikrokontroler

Makroima se simbolička imena registara povezuju sa fizičkim adresama na magistrali

```

13 .retain          ; Override ELF c
14                 ; and retain cur
15 .retainrefs     ; And retain any
16                 ; references to
17
18 -----
19 RESET          mov.w #_STACK_END,SP ; Initialize sta
20 StopWDT        mov.w #WDTPW|WDTHOLD,&WDTCTL ; Stop watchdog
21
22 -----
23 ; Main loop here
24
25 -----
26
27
28 -----
29
30 Stack Pointer definit
31 -----
32 .global _STACK_END
33 .sect .stack
34
35 -----
36 ; Interrupt Vectors
37 -----

```

```

msp430f5438.h
3726
3727 /*-----
3728 * WATCHDOG TIMER A
3729 *-----
3730 #define __MSP430_HAS_WDT_A__          /* Definitio
3731 #define __MSP430_BASEADDRESS_WDT_A__ 0x0150
3732 #define WDT_A_BASE                    __MSP430_BASEADDRESS_WDT_A__
3733
3734 SFR_16BIT(WDTCTL);                    /* Watchdog
3735 SFR_8BIT(WDTCTL_L);                  /* Watchdog
3736 SFR_8BIT(WDTCTL_H);                  /* Watchdog
3737 /* The bit names have been prefixed with "WDT" */
3738 /* WDTCTL Control Register
3739 #define WDTIS0
3740 #define WDTIS1
3741 #define WDTIS2
3742 #define WDTCNT
3743 #define WDTTMS
3744 #define WDTSSSEL
3745 #define WDTSSSEL
3746 #define WDTHOLD (0x0080) /* WDT - Tim
3747
3748 -----
3749
3750 -----
3751 #define WDTSSSEL0_L (0x0001) /* WDT - Tim
3752 #define WDTSSSEL0_H (0x0002) /* WDT - Tim
3753 #define WDTSSSEL1_L (0x0004) /* WDT - Tim
3754 #define WDTSSSEL1_H (0x0008) /* WDT - Tim
3755 #define WDTSSSEL2_L (0x0010) /* WDT - Tim
3756 #define WDTSSSEL2_H (0x0020) /* WDT - Tim
3757 #define WDTSSSEL3_L (0x0040) /* WDT - Tim
3758 #define WDTSSSEL3_H (0x0080) /* WDT - Tim
3759
3760 -----
3761 #define WDTPW (0x5A00)
3762 -----

```

Često korišćenim konstantama zadaju se simbolička imena

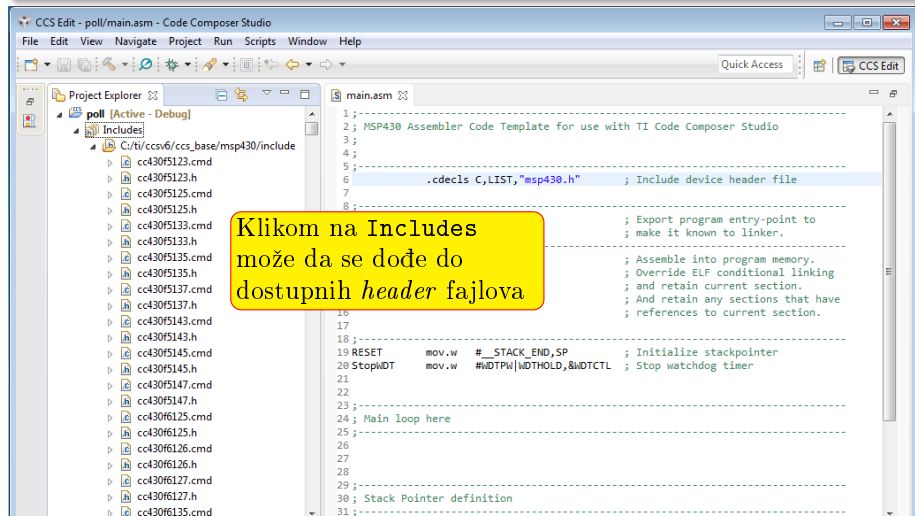
```

mov.w #WDTPW|WDTHOLD,&WDTCTL ; Stop watchdog
#define WDTPW (0x5A00)

```

# Hardverske specifičnosti 1/2

msp430.h je opšti *include* fajl za celu MSP430 familiju



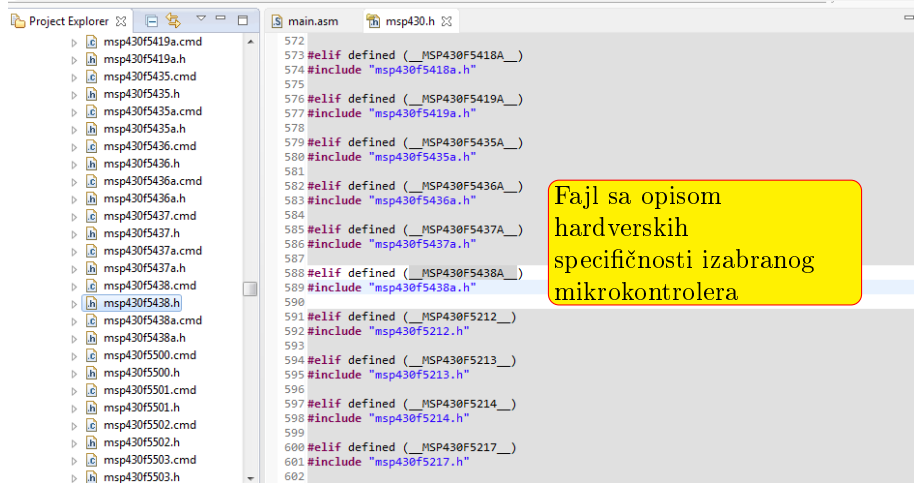
The screenshot shows the CCS Edit interface. On the left, the Project Explorer displays a tree view under 'poll [Active - Debug]' with an 'Includes' folder expanded to show a list of files in 'C:/ti/ccsv6/ccs\_base/msp430/include'. On the right, the main.asm file is open, showing assembly code with comments. A yellow callout box points to the 'Includes' folder in the Project Explorer.

**Klikom na Includes može da se dođe do dostupnih header fajlova**

```
1;-----  
2; MSP430 Assembler Code Template for use with TI Code Composer Studio  
3;  
4;  
5;-----  
6; .cdecls C,LIST,"msp430.h" ; Include device header file  
7;  
8;-----  
9; Export program entry-point to  
10; make it known to linker.  
11;-----  
12; Assemble into program memory.  
13; Override ELF conditional linking  
14; and retain current section.  
15; And retain any sections that have  
16; references to current section.  
17;-----  
18;-----  
19 RESET      mov.w  #_STACK_END,SP ; Initialize stackpointer  
20 StopWDT    mov.w  #WDTPH|WDTHOLD,&WDCTL ; Stop watchdog timer  
21  
22  
23;-----  
24; Main loop here  
25;-----  
26  
27  
28  
29;-----  
30; Stack Pointer definition  
31;-----
```

## Hardverske specifičnosti 2/2

Unutar fajla `msp430.h` definisanim makroima se uključuje fajl za izabrani mikrokontroler

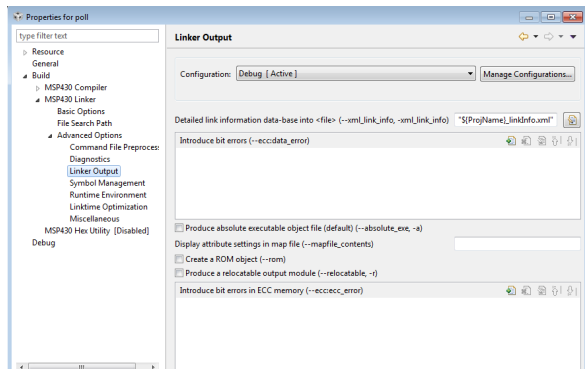


```
572
573 #elif defined (__MSP430F5418A__)
574 #include "msp430f5418a.h"
575
576 #elif defined (__MSP430F5419A__)
577 #include "msp430f5419a.h"
578
579 #elif defined (__MSP430F5435A__)
580 #include "msp430f5435a.h"
581
582 #elif defined (__MSP430F5436A__)
583 #include "msp430f5436a.h"
584
585 #elif defined (__MSP430F5437A__)
586 #include "msp430f5437a.h"
587
588 #elif defined (__MSP430F5438A__)
589 #include "msp430f5438a.h"
590
591 #elif defined (__MSP430F5212__)
592 #include "msp430f5212.h"
593
594 #elif defined (__MSP430F5213__)
595 #include "msp430f5213.h"
596
597 #elif defined (__MSP430F5214__)
598 #include "msp430f5214.h"
599
600 #elif defined (__MSP430F5217__)
601 #include "msp430f5217.h"
602
```

Fajl sa opisom hardverskih specifičnosti izabranog mikrokontrolera

# Linkovanje

Linkovanje je proces u kome se mašinski kod iz različitih asemblerskih fajlova prevodi u jedan jedinstven fajl koji se može spustiti u memoriju mikrokontrolera.



Najvažnije uputstva za linker se nalaze u *linker command file*-u

Ink\_msp430f5438a.cmd Segmenti mogu da se preklapaju

```

133 /* Specify the sections allocation into
134 .....*/
135
136 SECTIONS
137 {
138     .bss      : {} > RAM                /* Global & static vars
139     .data     : {} > RAM                /* Global & static vars
140     .TI.noinit : {} > RAM              /* For #pragma noinit
141     .sysmem   : {} > RAM              /* Dynamic memory allocation
142     .stack    : {} > RAM (HIGH)       /* Software system stack
143
144 #ifndef LARGE_DATA_MODEL
145     .text     : {} >> FLASH           /* Code
146 #else
147     .text     : {} >> FLASH2         /* Code
148 #endif
149     .text.isr : {} > FLASH           /* ISR Code space
150     .cinit    : {} > FLASH           /* Initialization tables
151 #ifndef LARGE_DATA_MODEL
152     .const    : {} > FLASH           /* Constant data
153 #else
154     .const    : {} > FLASH | FLASH2 /* Constant data
155 #endif
156     .cio      : {} > RAM             /* C I/O Buffer
157
158     .pinit    : {} > FLASH           /* C++ Constructor tables
159     .init_array : {} > FLASH         /* C++ Constructor tables
160     .mspabi.exidx : {} > FLASH      /* C++ Constructor tables
161     .mspabi.extab : {} > FLASH      /* C++ Constructor tables
162
163     .infoA    : {} > INFOA           /* MSP430 INFO FLASH Memory segments */
164     .infoB    : {} > INFOB
165     .infoC    : {} > INFOC
166     .infoD    : {} > INFOD
167
168     /* MSP430 Interrupt vectors
169     .int00     : {}                 > INT00
170     .int01     : {}                 > INT01

```

**.bss** je tip segmenta u koji se smeštaju neinicijalizovani podaci (RAM memorija)

**.data** je tip segmenta u koji se smeštaju inicijalizovani podaci (RAM memorija)

**.text** je tip segmenta u koji se smeštaju instrukcije (FLASH memorija)

**.const** je tip segmenta u koji se smeštaju konstantni podaci (FLASH memorija)

# Kompajliranje

Komanda BUILD vrši kompajliranje i linkovanje projekta

Nakon uspešnog kompajliranja klikom na ovu ikonu ulazi se u mod za debugovanje

Klik na ovu ikonu vrši kompajliranje i linkovanje celog projekta

```

25 ; Main loop here
26
27
28 setup
29
30          ; ulaznih bita
31          ; izlazni
32          ; biti iskluceni
33          ; koji se koristi
34          ; bit-u ulazni tako
35          ; da ih nema potrebe dodatno konfigurisati
36
37 loop    mov.b   P2IN,R9      ; učitava stanje porta P2 u registar R9
38        mov.b   R9,R8       ; cuvanje istog stanja u privremeno reg R8
39        bit.b   #0x00,R9    ; testira bit 7 porta 2, tj. taster S4
40        jz     ledon
41        bic.b   #0x08,P4OUT  isključuje diodu LD1
42        jmp    edge
43        ledon  bis.b   #0x08,P4OUT  ; uključuje diodu LD1
44        edge   xor.b   #0x00,R9    ; ako je bit 2 u R9 jednak 0 posle ove
45              ; operacije ce biti jednak 1
46        and.b   R7,R9        ; prebacivanje novog stanja porta u staro
47        mov.b   R8,R7       ; stanje za sledeci ciklus
48        bit.b   #0x00,R9    ; 1 na mestu bita 2 znaci da je detektovana
49        jz     loop
50

```

Nakon kompajliranja/linkovanja pojavljuju se pokazivači na eventualne greške i upozorenja

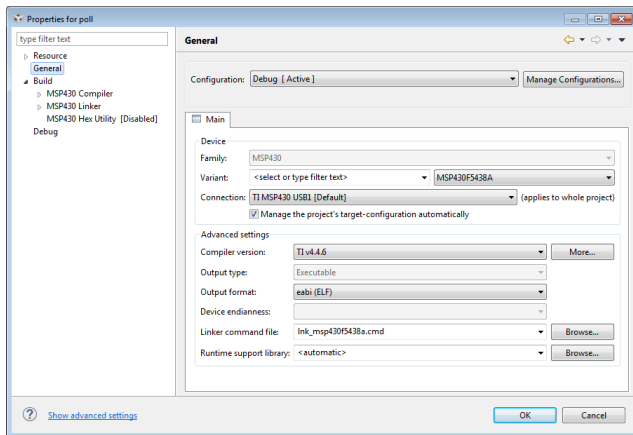
Description	Resource	Path	Location
Errors (1 item)			
[E0003] Unexpected trailing operand(s)	main.asm	/poll	line 39

Poruke koje ukazuju na status kompajliranja/linkovanja



# Debugovanje

Debugovanje je proces u kome se uz pomoć hardverskog debagera vrši testiranje aplikacije



The screenshot shows the CCS Debug interface with the following components and annotations:

- Run button:** A yellow callout box points to the green play button in the toolbar, stating: "Run - klik na ovo dugme izvršava program".
- Reset button:** A yellow callout box points to the red square button in the toolbar, stating: "Dugme za reset".
- Step Into button:** A yellow callout box points to the green square button in the toolbar, stating: "Step Into - klik na ovo dugme izvršava instrukciju koja je markirana zelenom".
- Breakpoint button:** A yellow callout box points to the blue square button in the toolbar, stating: "Klikom na ovo dugme prekida se debugovanje".
- Disassembly view:** A yellow callout box points to the disassembly window, stating: "Disassembly prozor prikazuje program onako kako je smešten u memoriju".
- Green marker:** A yellow callout box points to a green arrow on line 20 of the assembly code, stating: "Zeleni marker ukazuje na sledeću instrukciju koja treba da se izvrši".
- Disassembly line:** A yellow callout box points to line 37 of the disassembly, stating: "Svakoј liniji originalnog koda koja je prevedena u proces u kompajliranja odgovara jedna linija u Disassembly prozoru".

The assembly code in the main.asm window is as follows:

```

7
8
9
10
11
12
13
14
15
16
17
18
19
20 RESET mov.w # _STACK_END,SP ; Initialize stackpointer
21 StopMDT mov.w #MDTPM|MDTHOLD,&MDTCTL ; Stop watchdog timer
22
23
24 ; Main loop here
25
26
27
28 setup bis.b #0x78,P4DIR ; LE diode su na portu P4
29 bis.b #0x7f,P6DIR ; inicijalizacija odgovarajucih bits
30 ; portova P6 i P11 da budu ulazni
31 bic.b #0x7f,P6OUT ; inicijalno su svi segmenti iskluceni
32 bis.b #0x02,P11DIR ; aktivira se samo displej koji se koristi
33 bic.b #0x02,P11OUT ; svi portovi su po default-u ulazni tako
34 ; da ih nema potrebe dodatno konfigurisati
35 loop mov.b P2IN,R9 ; ucitava stanje porta P2 u register R9
36 mov.b R9,R8 ; cuvanje istog stanja u privremenom reg R8
37 bit.b #0x08,R9 ; testira bit 7 porta 2, tj. taster S4
38 jz ledon
39 bic.b #0x08,P4OUT ; iskljucuje diodu LD1
40 jmp edge

```

The disassembly view shows the following code:

```

005bfc: 1AE2 5C4C RPT SR RLAX.L R12
20 RESET mov.w # _STACK_END,SP ; Initialize stackp
$.:/main.asm:20:605(,), RESET():
005c00: 4031 5C00 MOV.W #0x5c00,SP
21 StopMDT mov.w #MDTPM|MDTHOLD,&MDTCTL ; Stop watchdog tim
StopMDT:
005c04: 40B2 5A80 015C MOV.W #0x5a80,&watchdog_Timer_MDTCTL
28 setup bis.b #0x78,P4DIR ; LE diode su na portu P4
setup:
005c0a: D0F0 0078 A617 BIS.B #0x0078,Port_3_4_P4DIR
; inicijalizacija odgovaraj
7f,Port_5_6_P6DIR
; inicijalno su svi segment
7f,Port_5_6_P6OUT
; aktivira se samo displej
rt_F_P6DIR
; svi portovi su po default
rt_F_PFOUT
; ucitava stanje porta P2 u
loop:
005c24: 4059 A5DB MOV.B Port_1_2_P2IN,R9
36 mov.b R9,R8 ; cuvanje istog stanja u pr
005c28: 4948 MOV.B R9,R8
005c2a: B079 0080 BIT.B #0x0080,R9
38 jz ledon
005c2e: 2403 JEQ (ledon)
39 bic.b #0x08,P4OUT ; iskljucuje diodu LD1
005c30: C2F0 A5F1 BIC.B #8,Port_3_4_P4OUT
40 jmp edge
005c34: 3C02 JMP (edge)
41 ledon bis.b #0x08,P4OUT ; ukljucuje diodu LD1

```

The screenshot displays the Code Composer Studio (CCS) interface. The View menu is open, showing various tool windows. A yellow callout box points to the 'Registers' window, which displays a table of core registers. Another yellow callout box points to the 'View' menu, and a third points to a blue marker in the assembly code.

**Prozor sa registrima**

Name	Value	Description
Core Registers		
PC	0x005C00	Core
SP	0x005C00	Core
SR	0x0000	Core
R3	0x000000	Core
R4	0x0C8D6D	Core
R5	0x00FFD0	Core
R6	0x000018	Core
R7	0x0000F8	Core

**Dugme za postavljanje Breakpoint-a**

**U View meniju se vrši izbor prikaza različitih resursa promenljivih/parametara**

**Plavi marker ukazuje na breakpoint**

```

22 setup    bis.b    #0x78,P4DIR    ; LE diode su na portu P4
29         bis.b    #0x2f,P6DIR
31         bic.b    #0x7f,P6OUT
32         bis.b    #0x02,P11DIR
33         bic.b    #0x02,P11OUT
34
35 loop    mov.b    P2IN,R9    ; ucitava stanje porta P2 u registar R9
36         mov.b    R9,R8    ; cuvanje istog stanja u privremenom reg R8
37         bit.b    #0x80,R9    ; testira bit 7 porta 2, tj. taster S4
38         jz      ledon
39         bic.b    #0x08,P4OUT    ; iskljucuje diodu LD1
40         jmp     edge
41 ledon    bis.b    #0x08,P4OUT    ; ukljucuje diodu LD1
  
```

# Debugovanje

Najjednostavniji i najpregledniji vid simulacije je *single stepping*

- Problem je što to može dugo da potraje i zamorno je

Mnogo efikasniji vid debugovanja je postavljanjem *Breakpoint*-a na dobra mesta

Generalno, *Breakpoint* se postavlja tamo gde treba simulirati spoljni događaj preko registra (u našem slučaju pritisak tastera) i tamo gde se očekuje reakcija na taj događaj

The screenshot shows the CCS Debug environment with the following components:

- Debugger Window:** Shows the current target as TI MSP430 USB1\_0/MSP430 (Suspended - HW Breakpoint) at main.asm:35:0x005C24. A red box highlights the Run button in the toolbar.
- Registers Window:** A table showing the state of various registers.
 

Name	Value	Description
0x00 P4OUT7	0	P4OUT7
0x00 P4OUT6	0	P4OUT6
0x00 P4OUT5	0	P4OUT5
0x00 P4OUT4	0	P4OUT4
0x00 P4OUT3	0	P4OUT3
0x00 P4OUT2	0	P4OUT2
0x00 P4OUT1	0	P4OUT1
0x00 P4OUT0	0	P4OUT0
0x78 P4DIR	0x78	Port 4 Direction [Memory Mapped]
- Source Code Window:** Shows assembly code for main.asm and function.asm. A red box highlights the instruction `mov.b P2IN,R9` at line 35, which is the main breakpoint.
- Annotations:** Three yellow callout boxes provide additional context:
  - Top-left: "Uz dobro osmišljen položaj *Breakpoint*-a testiranje se svodi na nekoliko uzastopnih pritisaka na "Run" ikonicu"
  - Center: "Kad god se izvršavanje programa zaustavi moguće je menjati sadržaj registra"
  - Bottom-right: "S obzirom da se testira da li program detektuje pritisak tastera ovo je dobar položaj drugog *breakpoint*-a"

# Pregled

- 1 Razvojno orkuženje
  - Hardversko razvojno okruženje
    - Portovi
  - Softversko razvojno okruženje
    - *Include* fajlovi
    - Linkovanje
    - Kompajliranje
    - Debagovanje
- 2 Primeri
  - Rad sa diodama
  - Rad sa tasterima
  - Prekidi
  - Sedmosegmentni displej

## Povezivanje dioda sa mikrokontrolerom

U primerima koje radimo na ovom kursu koristimo LE Diode. Zajedno sa LE diodama se redno vezuje otpornik koji služi za ograničenje struje diode (videti jednačinu za struju diode)

LE Dioda sa rednim otpornikom vezuje se na pin mikrokontrolera. Pin na koji je vezana dioda mora biti inicijalizovan kao izlazni. Da bi se dioda uključila potrebno je dovesti odgovarajući napon na jedan od krajeva diode.

Električna šema koja ilustruje način vezivanja diode prikazana je na nekom od prethodnih slajdova. Na osnovu prikazane šeme, koji logički nivo uključuje diodu?

# 1. Zadatak - Blinkanje diode

## Zadatak

Napisati program koji sa periodom od 1s uključuje i isključuje diodu na pinu 3 paralelnog porta 4.



# 1. Zadatak - Blinkanje diode

## Rešenje

Ovaj primer ilustruje upotrebu paralelnih portova kada se pinovi na portu koriste kao izlazni. Da bi se pinovi pravilno inicijalizovali i da bi se sistem ponašao na zahtevani način potrebno je uraditi sledeće:

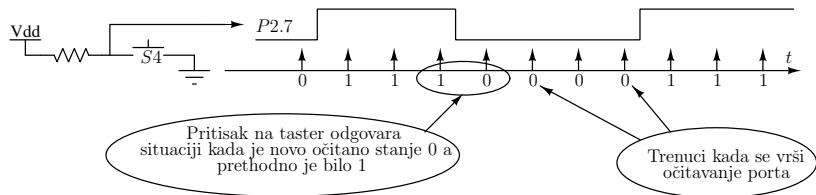
- Isključiti *watchdog* timer
- Podesiti pin kao izlazni
- Dodeliti pinu inicijalnu vrednost (U ovom primeru je svejedno koja je inicijalna vrednost)
- Menjati stanje diode sa periodom od 0.5s

## Kôd

Videti primer *v2-z1-blink* u materijalima

## Detekcija pritiska tastera

Algoritam detekcije pritiska tastera se zasniva na detekciji opadajuće ivice na ulaznom pin-u. Glavni program mikrokontrolera treba periodično da očitava stanje ulaznog pin-a, poredi ga sa vrednosti iz prethodnog očitavanja i u slučaju detekcije prelaska sa 1 na 0 izvrši zahtevanu radnju.



Taster se sa rednim otpornikom povezuje na pin mikrokontrolera. Potrebno je pin mikrokontrolera, na koji je vezan taster, inicijalizovati kao ulazni.

## 2. Zadatak - Promena stanja diode pritiskom na taster (Detekcija silazne ivice)

### Zadatak

Napisati program koji detektuje pritisak tastera S1 i kao rezultat detekcije menja stanje diode LD1.

## 2. Zadatak - Promena stanja diode pritiskom na taster

### Rešenje

Ovaj primer ilustruje upotrebu paralelnih portova kada se pinovi na portu koriste kao ulazni. Pored toga u ovom primeru je ilustrovan i jedan od softverskih načina za detekciju pritiska tastera (detekcija silazne ivice). Da bi se pinovi pravilno inicijalizovali i da bi se sistem ponašao na zahtevani način potrebno je uraditi sledeće:

- Isključiti *watchdog* timer
- Pin na koji je povezana dioda inicijalizovati kao izlazni
- Dioda inicijalno treba da bude isključena
- Inicijalizovati pin na koji je povezan taster kao ulazni pin
- U glavnoj programskoj petlji detektovati pritisak tastera na način objašnjen u nekom od prethodnih slajdova

### Kôd

Videti primer [2-z2-button-toogle](#) u materijalima

### 3. Zadatak - Promena stanja 4 diode (Detekcija nivoa signala)

#### Zadatak

Napisati program koji drži aktivnim sve 4 LE diode za vreme dok je taster pritisnut. Kada taster nije pritisnut diode su isključene.

### 3. Zadatak - Promena stanja 4 diode (Detekcija nivoa signala)

#### Rešenje

Ovaj primer ilustruje upotrebu paralelnih portova kada se pinovi na portu koriste kao ulazni. Pored toga ovaj primer ulustruje jedan od načina detekcije niva signala. Zbog načina na koji su tasteri povezani u kolo, nivo koji treba detektovati u ovom primeru je nivo "0".

Da bi se pinovi pravilno inicijalizovali i da bi se sistem ponašao na zahtevani način potrebno je sprovesti niz koraka kao u prethodnom primeru.

#### Kôd

Videti primer *v2-z3-button-glow* u materijalima

# Prekidi

Jedan od pristupa detekciji pritiska tastera jeste poliranje (stalno ispitivanje da li je došlo do promene stanja tastera). Ovakav pristup nije "efikasan" jer procesor troši dosta vremena na radnju koja se retko dešava.

Zašto mikrokontroler ne reaguje samo kada se pritisne taster?

Mehanizam koji omogućava ovakav pristup naziva se prekid.

Izvori prekida su mnogobrojni (UART, Timer, Paralelni port, DMA ,...)

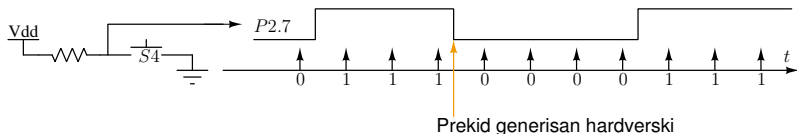
## Prekidi portova

Svaki pin na protovima P1 i P2 može da generiše prekid.

Konfiguracija prekida vrši se pomoću registara PxIFG, PxIE i PxIES

Bit u PxIFG registru se setuje kada se desi prekid na odgovarajućem pinu. Pri izlasku iz prekida neophodno je resetovati prethodno setovani bit!

Prekidi koji se generišu promenom stanja ulaznih pinova portova P1 i P2





## 4. Zadatak - Promena stanja diode

### Zadatak

Rešiti 2. zadatak koristeći mehanizam prekida

## 4. Zadatak - Promena stanja diode

### Rešenje

Ovaj primer ilustruje način obrade prekidnog zahteva u prekidnoj rutini. Da bi generisanje prekida bilo moguće potrebno je dozvoliti generisanje prekida i pravilno inicijalizovati pinove koji se koriste. U tu svrhu potrebno je uraditi sledeće:

(U glavnom programu)

- Isključiti *watchdog* timer
- Pin na koji je povezana dioda inicijalizovati kao izlazni i dioda je inicijalno isključena
- Inicijalizovati pin na koji je povezan taster kao ulazni pin i dozvoliti prekid na tom pinu (P2IE)
- Podesiti da se prekid generiše usled silazne ivice (P2IES)

## 4. Zadatak - Promena stanja diode

### Rešenje

Ovaj primer ilustruje način obrade prekidnog zahteva u prekidnoj rutini. Da bi generisanje prekida bilo moguće potrebno je dozvoliti generisanje prekida i pravilno inicijalizovati pinove koji se koriste. U tu svrhu potrebno je uraditi sledeće:

(U prekidnoj rutini)

- Na početku prekidne rutine proveriti izvor prekida (ispitivanje bita P2IFG registra)
- Promeniti stanje diode
- Resetovati odgovarajući bit P2IFG registra

### Kôd

Videti primeri *v2-z4-button-toggle-isr* u materijalima

## 5. Zadatak - Brojač

### Zadatak

Potrebno je napisati program koji realizuje funkcionalnost brojača. Vrednost brojača se uvećava za 1 pritiskom na taster S1 dok taster S2 umanjuje vrednost brojača za 1. Opseg vrednosti brojača je 0-255. Kada vrednost brojača izađe iz opsega brojač se "prevrti".

## 5. Zadatak - Brojač

### Rešenje

Ovaj primer ilustruje obradu dva prekidna zahteva koja dolaze od istog izvora prekida. Da bi generisanje prekida bilo moguće potrebno je dozvoliti generisanje prekida i pravilno inicijalizovati pinove koji se koriste. U tu svrhu potrebno je uraditi niz koraka kao u prethodnom primeru!

### Kôd

Videti primer *v2-z5-counter* u materijalima

## 6. Zadatak - Brojač po modulu 16

### Zadatak

Potrebno je napisati program koji realizuje funkcionalnost brojača. Vrednost brojača se uvećava za 1 pritiskom na taster S1 dok taster S2 umanjuje vrednost brojača za 1. Opseg vrednosti brojača je 0-15. Kada vrednost brojača izađe iz opsega brojač se "prevrti". Na sedmosegmentnom displeju (koji se nalazi na razvojnoj ploči) ispisuje se vrednost brojača u heksadecimalnom zapisu.

## 6. Zadatak - Brojač po modulu 16

### Rešenje

Ovaj primer ilustruje obradu dva prekidna zahteva koja dolaze od istog izvora prekida. Da bi generisanje prekida bilo moguće potrebno je dozvoliti generisanje prekida i pravilno inicijalizovati pinove koji se koriste. Pošto se vrednost brojača ispisuje na sedmosegmentnom displeju neophodno je ispravno inicijalizovati i pinove koji su povezani sa ovim displejom.

### Kôd

Videti primer *v2-z6-7seg-counter* u materijalima

Kako ispisati dvocifreni broj na displeju?